M-Clones : Multiclass CLOsed queueing Networks Exact Sampling

Christelle Rovetta

Inria Paris - Département d'informatique de l'ENS Paris UMR 8548

christelle.rovetta@inria.fr

Mots-clés : *Simulation parfaite, réseau fermé de files d'attente.*

1 Introduction

Ce résumé présente `M-Clones`, une bibliothèque Python permettant de réaliser la simulation parfaite [1] de réseaux fermés de files d'attente multi-classes décrits dans l'article de Bouillard et al. [2]. Nous présentons la problématique, l'idée principale introduite par [2] et enfin un rapide aperçu de l'implémentation de `M-Clones`. La bibliothèque `M-Clones` est disponible à l'adresse http ://www.di.ens.fr/∼rovetta/.

1.1 Problématique

Nous considérons un réseau fermé de K files d'attente $./M/1$ avec Z classes de clients. Pour chaque classe z, on note M_z le nombre de clients de la classe z et M le nombre total de clients. Nous supposons que les clients ne sont pas autorisés à changer de classe. Un état du réseau est caractérisé par le nombre de clients dans chaque file et chaque classe. Par exemple, pour le réseau représenté en figure 1, si $M_1 = 2$ et $M_2 = 3$ alors $x = ((1,0),(1,2),(0,0),(0,0),(0,1))$ est un état du réseau. L'espace de tous les états possibles est noté $\mathcal{S}$ et a une taille en $O(M^{KZ})$.

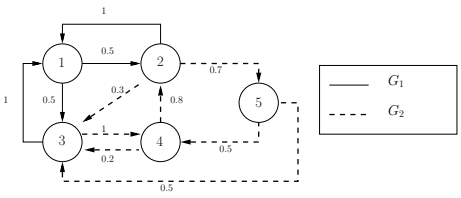

FIG. 1 – Réseau à 2 classes.

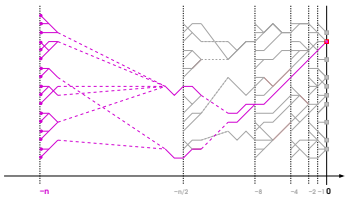

FIG. 2 – Simulation parfaite.

L'évolution du réseau est modélisée par une chaîne de Markov ergodique à valeur dans $\mathcal{S}$ [2]. On considère $t_{i,j,z} : \mathcal{S} \to \mathcal{S}$ la fonction qui réalise le départ d'un client de la classe z de la file i vers la file j. Les transitions de la chaîne se font en appliquant la fonction $t_{i,j,z}$. Pour chaque transition, un triplet (i,j,z) est choisi selon plusieurs critères. La file de départ i est tirée aléatoirement proportionnellement à son taux de service. La classe z est choisie en fonction d'une politique de service (classe prioritaire, plus grand nombre de clients, ...) qui ne dépend que du nombre de clients de chaque classe en file i. Enfin, la file d'arrivée j est tirée aléatoirement selon les probabilités de routage. Par exemple si on tire $i = 2$ et si la règle dans la file 2 est de servir la classe contenant le plus de clients alors le triplet choisi pour l'état `x` sera $(2,5,2)$ avec probabilité $0,7$.
L'algorithme de simulation parfaite [1] (voir figure 2) permet l'échantillonnage sans biais de la distribution stationnaire d'une chaîne de Markov ergodique. Le prix à payer est de devoir effectuer la simulation à partir de tous les états. Cet algorithme est donc irréalisable en pratique pour notre modèle car $|\mathcal{S}| = O(M^{KZ})$.

1.2 Représenter un ensemble d'états par un diagramme

L'idée introduite dans [2] est d'exploiter la contrainte sur le nombre de clients pour proposer une représentation de l'espace des états plus compacte, appelée diagramme.

Un diagramme est un graphe orienté représentant un ensemble d'états. Dans un diagramme, les états sont encodés par des chemins de longueur K (le nombre de files) partant du nœud $[0,(0,\ldots,0)]$ et arrivant au nœud $[K,(M_1,\ldots,M_Z)]$. L'arc $([k{-}1,(m_1,\ldots,m_Z)],[k,(m'_1,\ldots,m'_Z)])$ encode l'information suivante : «il existe un état dont le nombre de clients en file k pour chaque classe z est égal à $m'_z - m_z$ ». Les transitions $t_{i,j,z}$ sont directement effectuées sur le diagramme en $O(KM^{2Z})$.

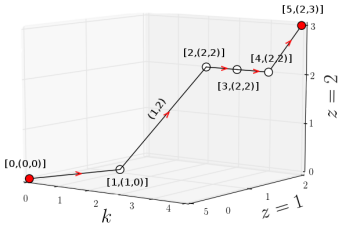

FIG. 3 – Diagramme représentant $\{x\}$.

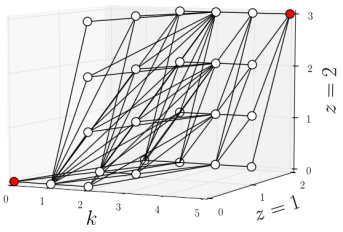

FIG. 4 – Diagramme représentant $\mathcal{S}$.

2 Implémentation en Python

Pour implémenter les algorithmes dans [2] nous avons choisi le paradigme orienté objet, ce qui a entraîné le choix du langage Python d'une part pour sa portabilité et d'autre part pour son ergonomie. La bibliothèque `M-Clones` contient 3 modules : `model` (pour la définition du modèle), `state` (pour manipuler les ensembles d'états comme des vecteurs) et `diagram`. Dans ce dernier figurent les classes `Diagram`, `Column` et `Arc` qui encodent respectivement un diagramme, un ensemble d'arcs pour un k donné et un arc. Plusieurs méthodes sont définies dans la classe `Diagram`, en voici les principales : `T` réalise une transition sur le diagramme, `reset` transforme le diagramme courant en un diagramme contenant tous les états, `psi` retourne l'ensemble d'états du diagramme, `plot` dessine le diagramme pour $Z \leq 3$ et `exactSample` utilise la simulation parfaite pour produire un état distribué selon la distribution stationnaire.

Un diagramme est encodé par un tableau de longueur K. Chaque case du tableau contient un objet de type `Column`. Les classes `Column` et `Arc` sont des métaclasses et héritent donc d'un `type` Python. Elles bénéficient ainsi des méthodes (nombreuses et optimisées) définies pour leur type. La classe `Column` hérite du type `set` et encode un ensemble d'objets de type `Arc` eux-mêmes définis à partir du type `tuple`.

Un tutoriel sous forme de TP dont le but est d'encoder un algorithme de simulation parfaite en utilisant les diagrammes est également disponible avec les sources de `M-Clones` [1].

Références

[1] J.G. Propp, D.B. Wilson Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random structures and Algorithms*, 1996

[2] A. Bouillard, A. Bušić, and C. Rovetta. Perfect sampling for multiclass closed queueing networks. In *12th International Conference on Quantitative Evaluation of SysTems, QEST 2015*, 2015.

1. Ces travaux ont été réalisés dans le cadre du projet ANR-12-MONU-0019.